Report No. 7164

More SURAP2 Hierarchical Routing Issues (SRNTN-34)

Gregory Lauer and Ross Callon

Prepared By:

BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02138

Prepared for:

DARPA/ISTO
1400 Wilson Bl.
Arlington, VA. 22209

Sponsored by:

The Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

This research is supported by the Information Processing Technologies Office of the Defense Advanced Research Projects Agency under contracts: MDA-903-83-C-0173 & N00140-87-C-8910. The views and conclusions contained in this document are those of the authors and do not represent the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the Army or the United States Government.

Contents

1. Introduction

This paper discusses a number of issues relating to hierarchical routing for Surap 2. It is assumed that Surap 2 will use a three level (PR units, clusters, and superclusters) hierarchical structure, with PROP style routing used between PR units within a cluster, and SPF routing used between clusters and superclusters. The reasons for these design choices are discussed in detail in a related paper [1]. Section 2 describes the maintenance of cluster membership information, with respect to initialization of the network, partitioning and reconnection of the network, and operation when a PR unit moves or a CH or SCH dies. Section 3 describes Node Tracker (NT) algorithms. Section 4 of this paper describes the broadcast algorithms for use in Surap 2.

The emphasis throughout this paper is on the various options available for protocol design, and the reasons for particular design choices. The specific details of protocol design and unambiguous specification of protocol operation are topics for future work.

2. Cluster and SuperCluster Membership

2.1 Basic Principles for Cluster Membership

As discussed in [1], the ability for a PR unit to simultaneously belong to more than one cluster is important to allow data to be correctly routed to mobile PR units. In addition, this ability increases the robustness in the case of primary cluster failure. Since each PR unit whose primary cluster has failed will already belong to another cluster, they can therefore immediately inform one of the other clusterheads that it is its new primary cluster. The clustering algorithm therefore allows the PR units to belong to up to k clusters simultaneously. The variable k may be set to 1, 2, or 3 when the network is configured, and must be the same for all PR units.

A PR unit's primary cluster is treated differently from its secondary clusters in two respects: (1) The primary cluster ID is used in the source address field of outgoing data; and (2) The NT algorithm returns the primary cluster as the PR unit's address. When the PR unit changes its primary cluster, its correspondent PR units find out about this change by examining the "source address" field of the arriving data units. This implies that PR units must continuously monitor this field on arriving data. In other respects, the multiple clusters to which a PR unit belong are not given special treatment. For example, the PR unit participates in the internal intra-cluster routing in all of its clusters.

In order to minimize the disruption caused by changing of primary clusters, it is desirable to pick as primary the cluster in which the PR unit will be likely to stay the longest. In addition, it is also desirable to not change primary cluster too often. For example, it is important that a single link going up and down intermittently be unlikely to cause a PR unit to rapidly oscillate between two alternative primary clusters. We propose that the PR units join the primary cluster corresponding to the closest clusterhead. In case of a tie, the PR unit will not change primary cluster (if it already belongs to one of the clusters in the tie) or will join one at random (if it is unclustered, or belongs to another cluster which is further away), while preferring a cluster which had already been a secondary cluster. This proposal ensures that although a link change which causes a path length to change by one unit could cause a PR unit to change primary clusters, it will take a larger change in path length to cause an oscillation in primary clusters. Although there is no guarantee that the PR unit will stay in this "closest" cluster the longest, in the absence of more detailed information about the likely future path of the PR unit this is probably the best choice.

When a PR unit changes its primary cluster, it will immediately notify its correspondent PR units, and its new primary CH will update the NT databases. It is possible that a relatively short time period (less than one PROP cycle) may elapse before packets begin to arrive addressed to that PR unit in its new cluster. For this reason, it is important that all of the PR units in the new primary cluster know a good route to the PR unit. This can only occur if this new primary cluster has already been a secondary cluster for some period of time. Similarly, as it may take some time for all of the correspondent PR units and NT address cache's to note the new address, it is also desirable that the old primary cluster remain a secondary cluster for a period of time (if packets arriving at an old cluster can be forwarded, then this is less critical, although still desirable). Finally, when a CH dies, the new primary CH will become the closest CH and thus become the primary. For this reason, it is desirable that the second closest CH be a secondary cluster.

It is not possible to design a clustering algorithm which guarantees to meet all of these desirable features. A straightforward, easy to implement, reasonable approximation is to have each PR unit join its *k* closest clusters, measured in terms of distance (number of hops) to the CH. Once again, in case of a tie, the PR unit will prefer clusters which were previously primary or secondary clusters, and will otherwise choose one at random.

By having all PR units join the closest clusters, we can ensure that each cluster is internally fully connected. For example, if PR units were to always continue to belong to a cluster for a fixed minimum time period after ceasing to have the cluster as their primary cluster, it is possible that a mobile PR unit could move so far away from its previous primary cluster that it still claimed to belong to it, even though none of its "good neighbor" PR units were members of the cluster. A more complex alternative would be to allow PR units to continue to belong to the old primary cluster for a fixed minimum time period only in the case where they continue to have a good path to the CH which lies entirely within the cluster. This alternative is not recommended because it seems unnecessarily complex, but could be considered in the future if observed network behavior warrants it. The possibility of having CHs request that PR units either leave or join their cluster is considered in Section 2.3.

2.2 Reporting Cluster Membership

It is important that the PR units report their cluster membership changes for node tracking and to allow the CHs to have the information needed to manage the cluster. This is done by sending a "PR State" Monitoring Data Packet (MDP) to each clusterhead. In the case that the PR unit has left a cluster, this packet is also sent to the old clusterhead. This packet indicates the primary and all secondary clusters, the reason why it is being sent, plus other information relevant to network monitoring.

The hierarchic routing table (HRT) is obtained from the neighboring PR units in the cluster just joined. This is done by having the PR, when it joins a cluster, indicate in its PROP that it has the no HRT, thus generating an update by the neighboring PRs. (See 4.3 for more details.)

2.3 Requesting and Refusing Cluster Membership

One proposal allows a CH to refuse membership in its cluster to a PR unit. This may be important, for example, in order to limit the cluster size, or to change the cluster shape in order to create a desired inter-cluster connectivity. A PR unit would therefore attempt to join the *k* closest clusters, and would end up belonging to the *k* closest that will accept it.

There are two ways to allow for this capability. With the most straightforward method, upon hearing about the presence of a nearby CH whose cluster it wants to join, a PR unit transmits a "request to join" to the CH. The CH would then respond with either the intercluster routes (ICR) packet, which would also imply acceptance, or would respond with a "cluster membership refused" packet. The problem with this method is that it complicates and slows down the normal process by which a PR unit joins clusters.

In the great majority of cases, the PR units will be allowed to join the closest clusters. It is therefore more desirable to implement a scheme which does not complicate this most frequent mode of operation.

The recommended second option therefore allows the PR unit to immediately join the closest clusters in the normal manner. When it joins a cluster, it sends a "PR state MDP" to the CH as described in section 2.2. If the CH does not want to have this PR unit in its cluster, it sends a "Please Leave" packet to the PR unit. The PR unit is required to obey this request. Similarly, a CH may send a "Please Join" packet to a PR unit which is adjacent to the cluster.

These packets would be used to give the network testbeds as much control over the logical structure of the network as is provided over its physical structure by PRISM. The number and type of packets required to control the logical structure of the network is currently under investigation.

2.4 An Overview of Supercluster Membership

Clusterheads join the supercluster associated with the nearest SCH (measured in terms of CH to CH hops, not PR unit hops). Each CH belongs to only one supercluster.

Network initialization is relatively straightforward. Initially, CHs find out about neighboring clusters through the PROPs. Clearly, the existence of a neighboring cluster implies the existence of a neighboring CH. CHs then exchange "clusterhead neighbor packets" (CNPs) with their neighboring CHs in order to determine the status of the interclusterhead link, and other important information (see below). Note that the neighboring cluster will not be in the HRT of the packet radios, and thus the CNPs must be delivered without using the HRTs. This requires that PRs be able to route to neighboring clusters without using the hierarchical routing algorithm.

There are two possible ways that CHs and SCHs can find out about each other. If a SCH is also a CH, then it (acting as a CH) can immediately join its own supercluster. It then indicates its supercluster membership to its neighboring CHs in the CNPs. If SCHs are not also CHs, then they must belong to one or more clusters organized by other CHs. In this case, the SCH has to announce its existence to each of its PR unit's clusterheads, again by use of the CNP.

The CNPs are exchanged between neighboring CHs, and can be triggered by any of three conditions: (1) when a CH first finds out about a neighbor CH; (2) when the CH state changes (for example, when a CH joins a supercluster); and (3) if more than x seconds have elapsed since the last CNP was sent. The CNP includes both the supercluster membership information, and a count of how far it is to the SCH. This allows CHs to determine the closest SCH, and therefore contains sufficient information to allow CHs to determine their supercluster membership. If in the future, we go to a system where the "width" of the inter-cluster path effects routing, the CNP will also give each CHs view of the link status.

As mentioned above, the "first" CH in the center of the supercluster (i.e., either the CH which is also a SCH, or any CH which has a SCH attached to a PR unit in its cluster) can initially find out about the SCH in a straightforward manner. This CH will then report its supercluster membership, and its distance to the SCH, in the CNP sent to each of its neighbor CHs. These neighbor CHs can then determine whether this new SCH is closer than their existing SCH. If so, then they join the new supercluster and remove themselves from the old cluster.

SCHs must know the identity of each CH in their supercluster. This may be determined by looking at the routing metric updates (RMUs) sent by each CH (see Section 2.5). Node Tracking information is determined by the periodic NT update broadcast messages sent by each CH. When a CH joins a new supercluster, it must obtain the corresponding supercluster routing table. This is accomplished in the same way (described in Section 2.5 below) that correction of formerly partitioned networks and network initialization are accomplished.

We may define additional packet types in the future, to support the superclusterhead obtaining more detailed information from the clusterheads. "Please Leave" and "Please Join" packets are defined in the same manner as for cluster membership.

2.5 Overview of SPF Operation

Clusterheads send out connectivity changes in "Routing Metric Update" (RMU) packets. Initially the SPF routing algorithm will use only connectivity in determining the intercluster routes. It is therefore unnecessary to send out updates when the length of the interclusterhead link changes. The RMU packets contain the status of all intercluster links with neighboring clusters, and also give the supercluster membership of all neighboring CHs.

RMUs can only be sent if (1) at least x seconds have elapsed since the last RMU was sent out; (2) if at least y seconds have elapsed since the CH last changed supercluster membership; or (3) if at least z seconds have elapsed since the CH was initialized. Point (1) prevents the CH from flooding the supercluster with an unlimited number of RMUs. Points (2) and (3) are useful to prevent an RMU from being sent out if changes are likely to occur almost immediately in the state of the CH or of its links to its neighbors. RMUs are broadcast throughout the supercluster using the algorithm recommended in Section 4.

In the case of initial network "startup", the delay of y seconds can greatly reduce the total RMU traffic sent out. Note that the CHs send out RMUs on an event driven basis. Thus, when they first join a supercluster, they would like to send out an RMU immediately. However, they also will be sending out CNP to their neighboring CHs. When the network is first initialized, it is quite likely that the neighbor CHs are either part of the "NULL" supercluster (i.e., they know of no SCH), or belong to a supercluster which is not as close as the newly discovered supercluster. The delay of y seconds is therefore set large enough to give each neighboring CH time to respond to the CNP before the RMU is sent out. Also when the network is first being brought up, if one SCH is turned on well before the others, then all CHs will at first belong to the same supercluster. In the presence of some initial fluctuations in supercluster membership, this could lead to a great deal of RMU traffic if there were not a specified maximum rate at which RMUs can be sent out.

When a new interclusterhead link is discovered, after the successful exchange of CNPs, the CHs must exchange routing information. In all cases, the new interclusterhead link will result in the broadcast of RMU packets from each CH giving its local connectivity. It is also necessary for the two CHs to send each other the complete network connectivity as they know it. This allows for correction of the routing information when two partitioned networks are joined, and during network initialization. The two CHs send each other the complete network connectivity as they know it by recreating the most recent RMU received from each peer node. This is possible because for routing the CH must know each peer node's connectivity and must store the most recent sequence number associated with each update.

In the case that these two CHs had previously been parts of separate partitions in a partitioned network, the RMUs will be broadcast by each receiving CH, and thereby allow each part of the network to correctly understand the other part's topology.

The RMUs are only necessary when the new neighbor CHs had no previous good path between them. However, even if the set of RMUs are always sent out, in the case where the two CHs had previously been part of a well connected network, the sequence numbers will all be correct and all messages will therefore be interpreted as old. This will keep the RMUs from being broadcast throughout the supercluster.

3. Node Tracker Algorithms

3.1 Requirements

PR units currently determine the identity of other PR units and devices in their cluster(s) by using the information contained in the PROPs. A Node Tracker (NT) algorithm is used to determine the address (including cluster and supercluster) for PR units and devices outside of the cluster, whether in the same supercluster, or in another supercluster. The mapping between devices and PR units is also provided.

Three types of packets may potentially be involved in the NT algorithm. "NT requests" are used to request the address and device-to-PR-unit mappings associated with a given device or PR unit. 'NT updates' may be used to transmit NT status information between CHs and/or SCHs. Finally, "NT responses" are returned in response to NT requests, and give the requested information. If it is necessary to execute a distributed search by broadcasting a NT request to multiple nodes (CHs or SCHs), this distributed search will be referred to as an 'NT search.'

We are assuming a three level hierarchy (PR units, CHs, and SCHs). NT functions are located at the CHs and SCHs. Individual PR units are not directly involved in the NT algorithm, except as originators of NT requests. The manner in which the PR units and devices notify their CHs of changes in cluster membership is discussed in Section 2.2.

The most difficult question in the design of the NT algorithm is how to distribute the NT information between the various components of the PR network. In particular, what is the distribution of responsibility between CHs and SCHs? Possible options are discussed in Section 3.2 below.

It is important to allow for the possibility of future changes to the NT algorithm without any resulting change to the PR units. Initially all NT requests from a PR unit will be sent directly to its primary CH. In the future, NT requests may instead be sent to the SCH associated with the primary CH. The PR units must therefore be configured so that the destination for the NT request can be set to either of these values without requiring a software change to the PR unit. In addition, the PR unit should not care about where the corresponding NT response is returned from. Depending upon the specific information requested, and what information is stored at the CH (or SCH), in some cases the CH may have the requested information. In other cases the CH may need to search for the information by requesting its SCH, or other CHs. However, if we were in the future to change the amount of information stored by the CH and/or SCH, or to change the NT algorithm used to find the NT information not stored at the CH or SCH, software changes would be required in the CHs and SCHs only. No change would be needed in the PR units.

3.2 Node Tracker Options

The "optimal" node tracker is the one which minimizes the amount of traffic associated with its operation, always responds with the correct address, and is simple to implement. It does not exist. For example, consider a network in which the nodes are static and there are many requests. Clearly the correct action is to have the NT database replicated in many locations, since we rarely have to update the database and this strategy minimizes the traffic associated with retrieving the address. On the other hand, consider a network in which there are very few requests for addresses and in which the nodes are constantly changing clusters. Clearly the correct action is to have the NT in a few locations or to require a search for the hierarchical address, since this minimizes the traffic associated with updating the database.

A reasonable goal for a node tracker design is that it produce a manageable amount of traffic for a wide range of update and request rates, responds with a useful address almost all the time and is not too difficult to implement.

To organize the discussion of node tracker options we consider 3 different aspects of node tracker operation. We first categorize them by describing what information is stored where. We then consider how quickly the node tracker must respond to requests for addresses and to updates. This step reduces the number of designs under consideration. Finally, we compare how much traffic is generate to update and access the node tracker for the remaining designs.

The various categories of node tracker algorithms are:

- At the clusterhead keep information about:
  1. the cluster
  2. the supercluster containing the clusterhead
  3. the entire network
- At the superclusterhead keep information about
  1. the supercluster
  2. the entire network

Now let us consider the response time requirements. There are two components to response time: that to updates, and that to requests. The former defines the probability of getting a correct response, the latter the speed with which a response is obtained. We note that due to the forwarding action taken by overlapping clusters, the response does not have to be completely accurate: if the address is a secondary cluster of the PR, then the message will be delivered successfully. We generally want to respond to a node tracker request rapidly; thus we can tolerate a node tracking system in which updates to the hierarchical address database are processed relatively slowly compared to the speed with which requests are answered.

A strategy which is consistent with the above observations is to send out updates to the NT database only periodically. If this interval is comparable to the time required for PRs to switch clusters, then the NT responses will generally be correct (returning a primary or secondary address) and the update traffic will have been greatly reduced. In the following analysis, we will always assume that updates (when required) are "bundled up" and sent out periodically.

In light of the above observations, we can quickly dispose of many of the node tracker options listed above. Since rapid response time to NT requests is important, algorithms which must search for the correct address are immediately at a disadvantage: any search will take a comparatively long time to perform. For example, if there is a uniform distribution of requests for addresses, nodes with only information about a supercluster or cluster will find that most of the requests require a search for the answer. In addition, since we cannot tolerate the delay associated with bundling up requests, each request can generate a search for an address; thus algorithms in which NTs do not keep track of the entire network will be slower and (potentially) generate more traffic. Another way of phrasing this is that, since we can limit the maximum amount of traffic generated by updates (via the bundling interval), we are most interested in algorithms which have the smallest amount of traffic associated with servicing a request.

Thus we will only consider algorithms which maintain a database of the entire network, either at the clusterhead or at the superclusterhead.

Before considering how much traffic is associated with these strategies, we must define the algorithms a little more.

3.3 Node Tracker at the CH

In this algorithm, every T seconds the CH sends the SCH a packet containing the changes that have occurred in the primary cluster membership (PR and device). (PRs automatically notify the CH of changes in cluster membership, so this information is already available at the clusterhead.) Every T seconds the SCH sends a packet to each other SCH describing the changes that have occurred in any of the clusters in its supercluster. In this manner each SCH is able to determine the primary hierarchical address of each device in the network. The superclusterhead then sends the changes in network hierarchical addresses to each of its clusterheads, thus maintaining the NT database at each CH. When PRs request a hierarchical address, the CH responds with the address in its NT database.

3.4 Node Tracker at the SCH

This algorithm is almost identical to the one above, it differs in that the SCH does not distribute the network hierarchical addresses to the CHs. Thus the CHs do not have the entire network database, only the SCHs do. The requests for an address must therefore be sent to the SCH for resolution, either directly by the PR or by the CH as agent for the PR. The response can be sent directly to the PR requesting the address or to the PR's primary CH, who forwards the response.

3.5 Node Tracker at the SCH with Caching at CH

As above, except the PR must ask the CH for the address and the SCH must send the response to the CH, who caches it for future reference.

3.6 Traffic Considerations

The algorithm which has the NT at the CH requires more traffic for updating the databases, since it requires the SCH to send the information to all its CHs. In addition, this algorithm requires the complete transmission of the NT database more often, since it must be transmitted to new CHs. However, this algorithm has the least possible traffic associated with requests, since they can all be answered by sending one packet to the closest CH. Thus this algorithm has the least "uncontrollable" traffic (since we can't control the rate at which requests are generated).

The algorithm with the NT at the SCH is has lower overhead, since it doesn't have to distribute the NT database to the CHs. Thus it will have less total traffic when the rate of NT requests is low enough. Sending the response directly to the PR is slightly more efficient than sending it to the CH, however it doesn't allow caching, and thus may end up generating more traffic (depending on the length of time the cache is valid and the number of "hits" on the cache).

Any of these algorithms ought to work well; we anticipate experimenting with them to determine how well each works.

4. Broadcast Algorithms for Surap 2

This Section describes the needs for broadcast algorithms in the Surap 2 design. The only broadcast required between PR units is the routing and direct connectivity information contained in the PROPs. The information sent from PR units to CHs, or from CHs to SCHs is done on a point-to-point basis. Therefore, this Section will discuss broadcast from a CH to each PR unit in the cluster, and between CHs and/or SCHs. The broadcast algorithms described here are in general used for routing information or NT requests (as described in each Section). Other details of NT design are discussed in Section 3.

4.1 Broadcast from Cluster Head to PR Unit

4.2 Requirements

There is a need for broadcasting an "Hierarchical Routing Table" (HRT) packet from each CH to the PR units in its cluster. The HRT packet gives the "next cluster" for each other cluster in the supercluster, and for each other supercluster. This is broadcast on an event driven basis, and supercedes all previous HRT packets.

4.3 Recommended Algorithm

This broadcast uses a relatively straightforward flooding of the cluster. The CH places the routing information in the HRT packet, along with a sequence number, and broadcasts it k times to its neighboring PR units (see discussion of k below). Each cluster uses an independent sequence number, chosen by the cluster head. Each PR unit, when it receives the packet, checks the sequence number. If this is an old packet, it is discarded. If the sequence number identifies a new HRT packet, the new information replaces the old table, and the PR unit broadcasts the packet k times.

The variable k will initially be set to one. If it is found to be common that some PR units fail to receive the broadcast, then k may be increased. Nonetheless, there will always be a small chance that a PR unit (or set of PR units) will fail to receive the broadcast. Therefore, in order to enhance the reliability of this method, the sequence number is included by each PR unit in each PROP. There are two possible ways to use this information:

1. Each PR unit, when it receives a PROP from a neighbor that shares its primary cluster, checks to see if its neighbor has old cluster routing tables. If so, it retransmits the HRT. Naturally, this implies that multiple neighbors are likely to be retransmitting the table, which should give a high probability of reception (although there may be an increased probability of hidden collisions in this case). In addition, this method implies that when the "slow" PR unit receives its new tables, it will

recognize the table as new, and will broadcast it as well. This may be helpful if there is a set of adjacent PR units that have failed to receive the routes. This approach is recommended due to its simplicity.

2. Each PR unit, when it receives a PROP from a neighbor that shares one or more common clusters, checks to see if it itself has old cluster routing tables. If so, it requests an update from the CH. The packet that it uses to request this information is the same that it uses to join the cluster (or change to primary or secondary status) in the first place (see Section 2.2). This method is not recommended, but is an acceptable alternative.

Note that a PR may, by examination of the PROPs, determine that its HRT is out of date. In this case, the HRT is flagged as old and never transmitted to other PRs. If a newer HRT is received, then it is accepted, but it is only marked new if it corresponds to the highest sequence number received.

If the recommended algorithm is used then, when a PR unit first joins a cluster, it may obtain an HRT from a neighbor by setting the sequence number to a predefined constant which indicates the absence of an HRT. This initial HRT packet serves a number of purposes, including initialization of both the inter-cluster (and supercluster) routing tables and the sequence number. Operation when a CH first comes up is discussed in Section 2.4.

4.4 Alternate Broadcast Algorithms

The above algorithm makes use of the fact that the most recent broadcast made all previous broadcasts obsolete. If we needed to broadcast other types of information within the cluster, we would need to augment the algorithm as follows.

This alternate algorithm is almost the same as the algorithm given above. The CH picks a sequence number and broadcasts k times. Each PR unit repeats the broadcast k times (where k = 1, 2, or 3). In this case however, since older messages are not obsoleted by the most recent, each PR unit remembers the last sequence number that it heard, plus any messages earlier than that sequence number that it hasn't gotten yet.

Once again, there is some chance that a PR unit (or set of PR units) will fail to receive a broadcast. Therefore, once again the sequence number (for each of its clusters) is included by each PR unit in each PROP. In this case, we may be missing the most recent message, and/or other earlier messages. It is not possible for a PR unit to determine that its neighbor is missing earlier messages on the basis of the sequence numbers in the PROPs alone. In this case, therefore, we have the PR units request retransmissions of the messages from the CH directly.

A possible third option for broadcasting within a cluster is to use reliable connections (e.g., SPP2) between adjacent PR units. This eliminates the need to add extra reliability by using the sequence number in the PROP. However, this eliminates the possibility of making use of the natural broadcast capability between PR units. This method is therefore not recommended.

4.5 Broadcast Between Clusterheads and/or Superclusterheads

The same broadcast methods will be used between CHs, between SCHs, and between CHs and SCHs. The requirements for this broadcast are discussed in Section 4.6. There are a large number of possible options that have been identified for this particular broadcast, with no obviously "best" option. For this reason, we will first discuss the possible options in Section 4.7, before recommending a particular option in Section 4.8.

The main differences between CH (or SCH) to CH (SCH) broadcast, as compared to broadcast within a cluster, are: (1) the lack of pure broadcast; (2) since the CHs are fewer in number, and greater in capability, we are less concerned about minimizing protocol complexity (e.g., we are more willing to have SPP2 connections between all neighboring CHs than between all neighboring PR units); and (3) The requirements for the type of data to broadcast differ slightly.

Throughout this Section, we have implicitly assumed that if reliable connections are required between CHs and/or SCHs, then SPP2 will be used for this purpose. This is not specifically intended as an endorsement of SPP2. Although this is the most likely choice in the Suran environment, other reliable end-to-end protocols, such as TCP, could be used instead. A complete evaluation of the various options for a reliable end-to-end protocol is beyond the scope of this document.

4.6 Requirements

Routing metrics for the SPF algorithm are broadcast by CHs to the other CHs in its supercluster, and by SCHs to all other SCHs. These Routing Metric Updates (RMUs) contain the complete local inter-cluster (or inter-supercluster) connectivity as seen by an individual CH (SCH). RMUs obsolete all previous similar messages from the same CH or SCH.

Routing Table Updates (RTUs) are broadcast from each SCH to all CHs in the supercluster. These contain the next supercluster to reach every other supercluster in the network.

These routing updates must initially be sent when the network is still being initialized. In particular, it may be expected that each CH (or SCH) knows the identity of its neighbors, but not that it knows the identities of all of its peer nodes, nor that it knows a route to all of its peer nodes. It is therefore required that the broadcast algorithm used for RMUs and RTUs work without knowledge of the complete routing tables.

Whether Node Tracker requests are to be done between CHs or between SCHs (or some combination of these) was discussed along with other aspects of NT requests in Section 3. Note that there is a question as to whether Node Tracker requests and updates should be structured so that each request/update obsoletes earlier ones. If this is done, then the broadcast algorithm is slightly simplified, and can be identical to the algorithm used for routing information. Even in this case, however, we must keep track of RMUs, RTUs, and NT packets separately, as a message of one sort clearly does not obsolete either of the other sorts.

NT requests are made only after the routing tables are available. Thus (1) it is possible (although not necessarily desirable) to use the routing tables in broadcasting NT requests; and (2) when the correct response is found to a NT request, this response may be returned as a normal point-to-point transmission.

4.7 Options

This Section describes a number of options for broadcast algorithms between CHs and/or SCHs. In order to more easily compare these algorithms, where necessary they will be illustrated by reference to the example network in Figure 1. Here there are six "nodes" over which a message is to be broadcast. These could be clusterheads within a supercluster, in which case the ovals in the Figure represent the clusters. Alternatively, the six nodes could be six superclusterheads, and the ovals would be the corresponding superclusters. The message to be broadcast originates at node A, and must be sent to the other five illustrated nodes B through E.

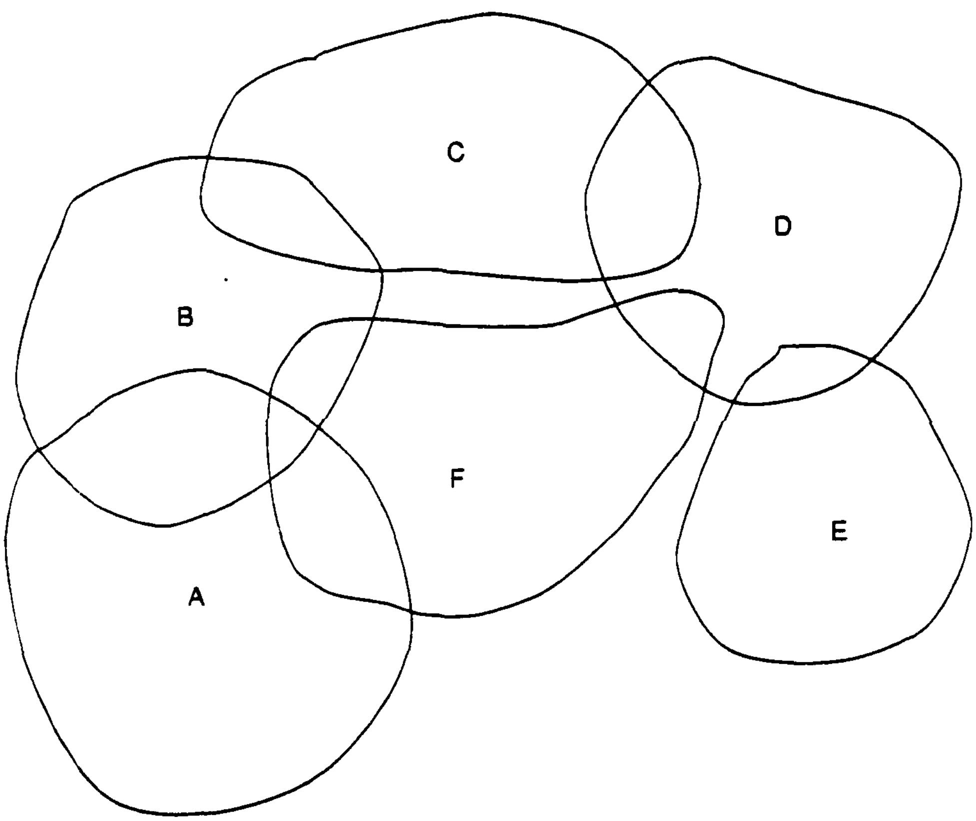

Figure 4.1: Example Network for Broadcast Algorithms

4.7.1 First Option; Spanning Tree Without Flooding

The first proposed algorithm is the one that we presented at the February 1985 designer's meeting in Palo Alto. This algorithm uses the intercluster routing table to create a spanning tree containing all cluster heads in the supercluster, without the need for flooding. Similarly the algorithm could be used to create a spanning tree containing all superclusterheads, using the inter-supercluster routing table. The requirement that the routing tables be available implies that this algorithm cannot be relied upon to distribute the routing information itself, but may be useful for Node Tracking.

The basic idea is that each message sent over any particular branch of the spanning tree contains a list of all destination nodes (CHs or SCHs) in that branch. When a node receives a message, it splits the message into several messages, each containing a subset of the destinations in the list that was received.

Suppose that node A has a broadcast message to transmit. It uses the inter-cluster or inter-supercluster routing tables to provide the identity of the other peer nodes, and a route to each of them. Note that it would be relatively simple to send a separate request to each other node (this is given as option 6, below). In some ways this may be more efficient than flooding the network with messages, but can be improved on as follows.

If cluster head A simply transmitted a separate message to each other node, this would result in nearly redundant requests on the links to nodes B and F. Thus, we propose that messages be piggybacked. Node A makes a list of all nodes to which it needs to send the message, determines the "next cluster" or "next supercluster" for each request, and then sends one message to the node for each "next cluster" or "next supercluster" that appears on the list. The message contains a list of all nodes to which this message is to be forwarded. Each node, when it receives the message, determines the appropriate "next cluster" or "next supercluster" for routing to each node on the list, and then forwards the message to the cluster head or superclusterhead of each of these.

This algorithm has advantages of being relatively simple, and minimizing the total traffic on the network. Unfortunately it fails if any node fails while holding the message. In addition, as it requires that routing tables be available, it cannot be used for routing messages.

4.7.2 Second Option; Augmented Spanning Tree Algorithm

A second option, therefore, is to augment this algorithm in order to eliminate this "single node" method of failure. There are two possible variations on this second option:

Each node could send the message to the next nodes on the tree as above, and then wait for each to respond before sending an acknowledgement to the previous node. This provides a definite acknowledgement to the source in the event of failure, but requires that each node keep track of the status of the message for an extended period of time.

Alternatively, each node could send the message on to the appropriate set of next nodes, and when it receives the SPP2 acknowledgements indicating that the next nodes have received the message, it notifies the previous node in the tree. Thus each node maintains a copy of the message until the next two levels of nodes receive the message. This method eliminates the "single failure point" problem of the first approach, but still suffers from the problem that it can only be used for NT traffic, not for routing messages. In addition, this method seems excessively complex for what it offers.

4.7.3 Third Option; Flooding with Acknowledgements

The third option is a fairly straightforward flooding method. The originating node picks a sequence number and broadcasts to all of its neighbors. Each node must remember the last sequence number used by every peer. Upon receipt of a broadcast message, each node checks the sequence number to see if this is a new message. If the message is new, the node sends and ACK to the neighbor who sent the message to it and broadcasts the message to each of its other neighbors. Otherwise, the node returns an ACK to the immediate neighbor who sent the message and otherwise ignores the message. After transmitting a message to each neighbor, the node must wait for either an ACK or the message in response. If these do not arrive within a timeout period, the transmission is repeated.

In our example, node A transmits a RMU message to nodes B and F. Node B returns an ACK to node A, updates its routing tables, and sends copies of the message to nodes C and F. Similarly node F sends an ACK to node A, updates its tables, and sends copies to nodes B and D. In this case, it is likely that the messages from B to F and from F to B will cross in mid-stream. In this case, there is no need for B and F to exchange acknowledgements. If either of these transmissions are lost in transit, then the sending node will not know that it was lost, as it thinks that the message going the other way implied acknowledgement. In this case, the node that received neither the transmission nor an acknowledgement over a link will retransmit the message. For this reason, if a node receives a duplicate message over a link it must acknowledge the duplicate. In our example, node D will act upon the first message received (either from C or F). The second message received at D will be ignored, except that an ACK will be returned to prevent retransmission.

4.7.4 Fourth Option; Flooding Over Reliable Links

The fourth option is the same as the third, except that the simple ACK is replaced by the use of a reliable protocol (such as SPP2). SPP2 only assures us of local delivery. The sequence number is still needed to damp out the broadcast. Here, upon receipt of a message, the node looks to see if it is new. If new, the message is sent to each neighbor node except for the one that the message is received from. If this is an old message, then no further action is necessary. Note that the acknowledgement scheme is handled by SPP2, and therefore no additional acknowledgements are necessary.

In our example, the operation is the same except that the acknowledgements are handled by the reliable (SPP2) protocol. If the messages from B to F and from F to B cross in mid-stream, SPP2 will nonetheless acknowledge each separately as no relation between the two will be known by the SPP2 implementations.

Option 4 differs from option 3 primarily in two aspects: (1) as discussed above, option 3 is slightly more efficient in the case of colliding messages (2) Method 4 has the significant advantage of using an already implemented (and fully debugged) protocol rather than a not-implemented (and possibly not fully thought-out) acknowledgement scheme.

4.7.5 Fifth Option; Extended Flooding Algorithm

The fifth option is a proposal by Nachum Shacham intended for Node Tracker requests and responses[2]. This method is an extension to the flooding method, which allows the NT response to be returned even in the event that the routing tables are not available, and offers a definite indication of failure when the NT request cannot be satisfied. In addition, a STOP message is described which is intended to reduce the extent of the search when a the NT requests has been satisfied. This method will only be very briefly described here. For a complete description, please see the original paper[2].

Basically, this method is the same as method 4 above (flooding over reliable links) with a number of additions. When each node first receives a search message, it remembers the node that sent the message to it as its "preferred" neighbor. Note that if you conceptually draw the link between each node and its preferred neighbor, you produce a spanning tree. When each node either finds the correct response to the NT request, or determines that its subtree (for which it is the root) does not contain the response, then it sends an appropriate message to its preferred neighbor. When the correct response is found, the response is sent by each node to its preferred neighbor, and therefore passed back to the original source. The node finding the correct response immediately sends a STOP message to all of its neighbors, which it turn broadcast the STOP message so as to flood the network with STOP messages until they catch up with the SEARCH message, thereby preventing additional searching. In addition, there are provisions for correct operation of the algorithm if the network topology should change during a search.

We have several reservations about this method. This method is considerably more complex than the simple flooding methods, and does not clearly offer significant advantages in our case. This method require that each node remembers significant data about each message being broadcast (more than just the sequence number of the most recent message) for a significant period of time. In times of network stress, there may be multiple simultaneous outstanding messages from different source nodes. This implies that there may be an excessive amount of data being maintained in order to support this algorithm. In the event of topological changes, if the answer is not found, then the method proposed to deal with these topological changes may require that the algorithm be re-run. This would add to the traffic and memory problems in high-stress situations.

For NT requests, we don't believe that the "STOP" message will significantly reduce the total traffic generated. For example, if we randomly choose the source of the request and the location of the item being searched for (the easiest case to analyze, and a probable "average" case), then on the average at least half of the network will have received the request by the time the STOP message can be initially transmitted (more if there are simultaneous arrivals). If the STOP message propagates no more quickly than the original request, then it will probably not stop much of the additional half of the network from receiving the request. If, on the other hand, the request propagates slowly (either intentionally to allow the STOP message to catch up, or inadvertently), then the memory requirements discussed above will be even more excessive. Even if the STOP message were to propagate instantaneously, the additional traffic generated by the STOP message would offset the gains made in limiting the transmission of the search.

4.7.6 Sixth Option; Repeated Transmissions

The sixth and final option is the simplest: the source node creates a separate message for each other node, and sends each message out separately. Although this may seem like a rather expensive option, for small networks it is actually quite efficient. For example, in the network in Section 4.7 (on page 14), node A would need to transmit five copies of the message. Two of these would require a single "big-hop" (i.e., hops between CHs or SCHs), two would require two big-hops, and one would require three big-hops (a total of 9 big-hops). In contrast, the flooding algorithm described as the fourth option would require that each node transmit a single message to a one of its neighbors. This would require a total of 9 transmissions (all between immediate neighbor nodes). Thus there would be fewer total packets, traversing roughly the same total distances (measured in big-hops between CHs or SCHs) through the network.

Consider networks of a constant average connectivity of C, and a total number of nodes N. As N increases, the average path length will increase roughly proportional to $\sqrt{N}$. As a rough estimate, we can figure the diameter of the network to be approximately equal to the square root of N, and the average path length to be approximately half this amount. The total number of packets with this method is equal to $N - 1$, thus the total distance traversed by these packets is approximately $(N - 1) \times \frac{\sqrt{N}}{2}$. In contrast, the total number of packets with option 4 is $N \times (C - 1) + 1$, each of which traverses a single "big-hop". All of these figures ignore the additional acknowledgement traffic generated by SPP2, as this will be proportional to the traffic mentioned here. The resulting traffic for networks of connectivity of 6 or of 10 is illustrated in Table 4.1.

N	C	Option 4	Option 6
49	6	168	246
64	6	252	321
100	6	495	501
144	6	858	721
196	6	1,365	981
64	10	252	577
100	10	495	901
196	10	1,365	1,765
289	10	2,448	2,602
400	10	3,990	3,601

Table 4.1: Number of Transmissions needed to Broadcast an Update by Flooding (Option 4) and by Repeated Transmission (Option 6)

In Table 4.1, the network size N is the number of nodes (i.e., the number of CHs in a SCH, or the number of SCHs in the network). Similarly, the "distance" is measured in big-hops. Table 4.1 shows clearly that for an average connectivity of 6 or larger, simply sending the message separately to each peer node is more efficient than a flooding algorithm until the network size becomes quite large. For an average connectivity of 6, the break-even point requires greater than 100 nodes in the network. For an average connectivity of 10, the break-even point is even larger. The two methods were roughly equal in our example network as the small network size was offset by the low average connectivity.

The main problem with simply sending out the message independently to each peer node is that this method, once again, requires that the routing tables be available in order to transmit the messages. This implies that this option cannot be used for routing information. For NT updates and requests, this option has several advantages. If we return a NAK (separate from the operation of any underlying reliable protocol) when the information is not found, then this algorithm would allow us to determine if the desired information is not available anywhere in the network. It is possible with this broadcast option to search through only part of the network. For example, if we are searching for a PR unit or device which was last known to be in a particular supercluster, we could look only in that supercluster and in it's neighboring superclusters. If the address is found before the entire network is searched, then the rest of the search can be eliminated.

4.8 Summary and Recommendation

The first, second and sixth methods proposed above require that the routing tables be available before they can be used. They are therefore not appropriate for broadcast of routing information (RMUs and RTUs). Routing information must be sent by a method that requires that each node know a route to its immediate neighbor nodes only. The fifth method offers three advantages that are relevant only to Node Tracker searching and has no advantage relative to the flooding methods for routing information, in spite of considerable additional complexity. For these reasons, we strongly recommend one of the two simple flooding methods (options 3 and 4) for dissemination of all routing information. Of the two, method 4 (flooding over reliable data paths) is preferred. The implementation and testing advantages of not having to implement an additional acknowledgement scheme is considered to outweigh the slight inefficiency of having to acknowledge both transmissions in the case where two nodes simultaneously broadcast the same message to each other.

For Node Tracker traffic, there may be implementation savings in being able to use the same broadcast algorithm as is used for the routing traffic. In addition, the "single point of failure" problem with method 1 is considered to make it unattractive. Methods 2 and 5 are both considerably more complex than the simple flooding methods. In addition, of the advantages offered by method 5, the first advantage (offering a return path for the successful response) is not necessary in our case since the normal routing tables will be available. The second advantage (giving a definite failure indication when the NT request cannot be satisfied, rather than waiting for a timeout) is not considered important. The third alleged advantage (the abbreviation of the search tree by the STOP message) is not believed to help at all. For these reasons, methods 1, 2, and 5 are all not recommended. The relative advantages of methods 3 and 4 remain the same. The choice therefore comes down to either method 4 or 6. In this case, because method 6 is so simple to implement, the advantage of using the same broadcast algorithm for NT traffic and routing traffic is not considered significant. For these reasons, we recommend option 6 for NT traffic.

Bibliography

[1] R. Callon and G. Lauer. *Hierarchical Routing for Packet Radio Networks*. Technical Report SRNTN-31, BBN Laboratories, Inc., June 1985.

[2] N. Shacham. *A Distributed Search Protocol for Large Networks*. Technical Report SRNTN-8, SRI International, Nov 1983.